

SYSTEM AND METHOD FOR NETWORK CACHING

BACKGROUND OF THE INVENTION

1. Related Applications.

The present invention claims priority from U.S.
5 Provisional Patent Application No. 60/197,490 entitled
CONDUCTOR GATEWAY filed on April 17, 2000.

2. Field of the Invention.

The present invention relates, in general, to network
information access and, more particularly, to software,
10 systems and methods for caching request and/or response
traffic in a data communication system.

3. Relevant Background.

Increasingly, business data processing systems,
entertainment systems, and personal communications systems
15 are implemented by computers across networks that are
interconnected by internetworks (e.g., the Internet). The
Internet is rapidly emerging as the preferred system for
distributing and exchanging data. Data exchanges support
applications including electronic commerce, broadcast and
20 multicast messaging, videoconferencing, gaming, and the
like.

Currently, Internet services are implemented as
client-server systems. The client is typically
implemented as a web browser application executing on a
25 network-connected workstation or personal computer,
although mail, news, file transfer and other Internet

services are relatively common. The server is typically implemented as a web server at a fixed network address. A client enters a uniform resource locator (URL) or selects a link pointing to a URL where the URL identifies the server and particular content from the server that is desired. The client request traverses the network to be received by the server.

The server then obtains data necessary to compose a response to the client request. For example, the response may comprise a hypertext markup language (HTML) document in a web-based application. HTML and other markup language documents comprise text, graphics, active components, as well as references to files and resources at other servers. In the case of static web pages, the web server may simply retrieve the page from a file system, and send it in an HTTP response packet using conventional TCP/IP protocols and interfaces. In the case of dynamically generated pages, the web server obtains data necessary to generate a responsive page, typically through one or more database accesses. The web server then generates a page, typically a markup language document, that incorporates the retrieved data. Once generated, the web server sends the dynamic page in a manner similar to a static page.

Many web sites have pages or page elements that are viewed by many users, and appear the same to all viewers. These page elements include data in the form of text files, markup language documents, graphics files and the like, as well as active content such as scripts, applets, active controls and the like. It is an inefficient use of bandwidth and server resources to resend the exact same data time after time from the server to different users.

Caching is designed to store copies of pages or page elements in caches located closer to the clients that are requesting the pages. When a web page can be served out of a cache, the page is returned more quickly to the user.

5 Caching reduces Internet traffic, for example, by storing pages at the Internet service provider (ISP) the first time a user accesses the pages so that subsequent requests for the same page do not require that the origin server be contacted. Another advantage of caching is that because

10 fewer requests reach the origin server, the origin server load is reduced for a given number of users.

A key issue in systems that cache Internet resources is control over the caching behavior. Conventional Internet caching methods allow a site owner to indicate

15 desired caching behavior by specifying cache control parameters in the HTTP packet headers. While HTTP v1.1 specifications, defined in IETF RFC 2068, allow cache parameters, many other network protocols do not offer these features. Also, while a web site owner does not

20 have actual control over the many caching mechanisms in the Internet and so cannot be positive that the desired caching behavior will in fact be implemented at each and every cache. As a result, many site owners simply specify all web pages as non-cacheable to minimize risks

25 associated with delivery of stale content, thereby forgoing the benefits of caching. A need exists for a system and method that provides for caching under control of rules specified by site owners.

Another limitation of existing Internet caching

30 mechanisms is they lack effective means that enable a site owner to control the contents of caches. Most cache systems are passive response caches that will cache a copy of a response to a given request, but will not place

content in a cache before a specific request. In some applications, however, performance can be improved if the web site could controllably load the cache by effectively pushing content from the web server out to the cache. For example, when a server is under heavy load, it may be desirable to expand the cache contents to explicitly prevent user requests from hitting the central server. Hence, a need exists for systems and methods enabling web site owners to actively manage network caches.

10 When the content being cached includes large files, it may take a significant amount of time to communicate the response and fill the cache. Examples include multimedia files, multicast or broadcast files, software updates, and the like. In these cases, the likelihood that a subsequent request for the same content will come in while the initial request is being filled is greater. Existing cache solutions will continue to forward requests to the server until the entire set of static files is cached so that the benefit of the cache is non-optimal. 15 Particularly in situations where large files (e.g., streaming video, media, software updates, and the like) it is contemplated that substantially simultaneous requests for the same content will be frequent. Accordingly, a need exists for a cache system and method that performs request merging to postpone requests while a prior request for the same content is pending or "in-flight". 20 25

Existing cache solutions are asynchronous in that the cache mechanism is unaware of operational and timing details within the origin server that might affect cache performance. For example, a particular web page might be updated within the origin server every minute or every half-hour. In such cases, the freshest content is obtained just after the origin server update, however, 30

because the cache does not know the update cycle, it cannot adjust the caching behavior to maximize the delivery of fresh content.

SUMMARY OF THE INVENTION

5 Briefly stated, the present invention involves a system for caching network resources in an intermediary server topologically located between a client and a server in a network. The intermediate server preferably caches at both a back-end location and a front-end location.
10 Intermediary server includes a cache and methods for loading content into the cache as according to rules specified by a site owner. Optionally, content can be proactively loaded into the cache to include content not yet requested. In another option, requests can be held at
15 the cache when a prior request for similar content is pending.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a general distributed computing environment in which the present invention is implemented;

20 FIG. 2A shows in block-diagram form significant components of a system in accordance with the present invention;

FIG. 2B shows in block-diagram form significant components of an alternative system in accordance with the
25 present invention;

FIG. 3 shows a domain name system used in an implementation of the present invention;

FIG. 4A shows components of FIG. 2A in greater detail;

FIG. 4B illustrates components of FIG. 2B in greater detail; and

5 FIG. 5 shows back-end components of FIG. 2B in greater detail.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

10 The present invention involves systems and methods for providing improved performance in network communications through the use of a caching web server located at the client-side "edge" of a network environment. This means that the web server implementing the cache in accordance with the present invention is logically proximate to the client application generating requests for web pages. It is contemplated that at least some of the requested content can be supplied by content data and resources persistently stored in a data store accessible by the web server, but that at least some content will be obtained from remote network resources such as remote data stores, remote web servers, and the like. This remote data is obtained by the web server on behalf of a requesting client and then cached so that it can be used to respond to subsequent requests. In a sense, the web server acts as a hybrid between a web server and a caching proxy server in that it can supply both original content as well as cached remote content.

20 The present invention is illustrated and described in terms of a distributed computing environment such as an enterprise computing system using public communication channels such as the Internet. However, an important feature of the present invention is that it is readily

5 scaled upwardly and downwardly to meet the needs of a particular application. Accordingly, unless specified to the contrary, the present invention is applicable to significantly larger, more complex network environments, including wireless network environments, as well as small network environments such as conventional LAN systems.

10 Essentially, an intermediary server(s) is placed in communication with the client and server to participate in the request/response traffic between the client and server. In this position, the intermediary can be given specific knowledge of the configuration, capabilities and preferred formats of both the clients and servers. The intermediary implements a cache for storing content including web pages, web page components, files, graphics, 15 program code and the like that are subject to client requests. Moreover, an intermediary server can modify the data that is subject to a client request and cache the modified data. For example, multimedia files may be formatted to a particular standard, such as an MPEG 20 standard, and cached in the modified form such that subsequent requests can be filled from cache without required formatting processes.

25 Although the present invention may use conventional cache instructions provided in HTTP headers, preferably the content includes caching parameters that provides independent instruction to the caching system of the present invention. In this manner, the caches of the present invention may provide caching services even where the content is marked uncacheable in the HTTP header.

30 Preferably, the cache contents are determinable by the site owner by allowing the site owner to cause the cache to be loaded with desired content. In this manner, content can be loaded into a cache before it is requested

by a user based on a likelihood that it will be requested by a user. In this manner, a web site owner can explicitly prevent hits to the central server(s) by proactively sending content to a cache. For example, in the case of a small web site of a few megabytes of storage, the first hit to an index page may justify sending the entire site to the intermediary server that received the hit thereby preventing any further hits to the origin server.

Moreover, the home page or "index.html" page of the web site (i.e., the page most likely to be the first contact with a user) may be cached on the intermediary permanently or until the website provides an updated index page. A hit to the index page may result in serving the index page from the cache, and a request from the intermediary server to load the remainder of the site, the linked pages, or an intermediate set of pages into the cache while the user views the index.html page.

In general, the present invention involves a combination of passive caching (i.e., cache contents are selected based on prior requests) and active caching (i.e., cache contents are selected speculatively based on what is expected to be requested in the future). Speculative caching is used in a variety of environments, such as disk caching, where data has a predictable spatial relationship with other data. However, in a network environment, particularly in a hyper-linked environment, the spatial relationships cannot be relied upon. HTML, for example, is specifically designed to access data in a non-sequential, user-controlled manner.

Accordingly, the present invention provides a system and method in which a current request, or a response to that request, can be used as a basis for both passively

and actively caching network resources associated with that request/response. A response can be passively cached in a conventional manner. Additionally, however, the request and/or response can be analyzed to determine other material that can be actively cached, such as data linked to the response.

This additional data may include dynamically generated data. For example, a current request may include state information such as a cookie, that is intended to allow the origin server to personalize responses. The cache may be filled by making speculative requests to the server using the cookie so that the cache is filled with dynamically generated content. Alternatively, state information can be communicated from the server in the initial response in the form of parameters encoded in links within the response. The cache may be filled by making speculative requests using the links within the current response so that the cache is filled with dynamically generated content.

In this manner, a cache can be intelligently and efficiently populated with data that has a high likelihood of being requested in the future even though that data may be stored in a variety of files and servers. Hence, the present invention provides a means to explicitly prevent accesses to the origin server by moving content into the cache speculatively, before it is requested.

Alternatively, speculative caching may be based on server resources. A server under load may become less able to provide certain types of resources. For example, an e-commerce web server under a high shopping load may dedicate a large quantity of resources (i.e., I/O buffers and memory) to shopping cart processes. This may make resources for other activities (e.g., serving

informational pages) scarce. In accordance with the present invention, an intermediary server is aware of server load and speculatively caches data that might otherwise tax scarce resources in the server itself.

5 While the examples herein largely involve web-based applications using markup language documents, web browsers and web servers, the teachings are readily extended to other types of client-server information exchange. For example, a database system can respond to an initial query
10 by loading the entire database or a selected portion of a database into the cache of an intermediary server. Subsequent requests can be satisfied by the intermediary server. Likewise, a network file system (NFS) or file transfer protocol (FTP) system may respond to an initial
15 hit for a top level directory by loading the cache of an intermediary server with files from all the branches of the directory or with a list of details for the files in the immediate subdirectories. These implementations are readily derived equivalents of the examples given herein.

20 In a particular implementation, the intermediary server is implemented by a front-end computer and a back-end computer that are coupled over a network. This enables either or both of the front-end and back-end computers to perform the caching functions as needed. In
25 this configuration, a back-end cache will cache responses to multiple front-ends. Conversely, the front-end caches will cache responses received from multiple back-ends. Hence, each front-end and back-end cache is uniquely situated within the network topology to improve cache
30 performance. For example, a back-end cache can readily identify cache contents with high hit rates and propagate that content to all or selected front-end caches. This

will result in requests being filled in the front-end caches before they reach the back-ends.

FIG. 1 shows an exemplary computing environment 100 in which the present invention may be implemented.

5 Environment 100 includes a plurality of local networks such as Ethernet network 102, FDDI network 103 and Token Ring network 104. Essentially, a number of computing devices and groups of devices are interconnected through a network 101. For example, local networks 102, 103 and 104
10 are each coupled to network 101 through routers 109. LANs 102, 103 and 104 may be implemented using any available topology and may implement one or more server technologies including, for example UNIX, Novell, or Windows NT networks, or peer-to-peer type network. Each network will
15 include distributed storage implemented in each device and typically includes some mass storage device coupled to or managed by a server computer. Network 101 comprises, for example, a public network such as the Internet or another network mechanism such as a fibre channel fabric or
20 conventional WAN technologies.

Local networks 102, 103 and 104 include one or more network appliances 107. One or more network appliances 107 may be configured as an application and/or file server. Each local network 102, 103 and 104 may include a
25 number of shared devices (not shown) such as printers, file servers, mass storage and the like. Similarly, devices 111 may be shared through network 101 to provide application and file services, directory services, printing, storage, and the like. Routers 109 provide a
30 physical connection between the various devices through network 101. Routers 109 may implement desired access and security protocols to manage access through network 101.

Network appliances 107 may also couple to network 101 through public switched telephone network 108 using copper or wireless connection technology. In a typical environment, an Internet service provider 106 supports a connection to network 101 as well as PSTN 108 connections to network appliances 107.

Network appliances 107 may be implemented as any kind of network appliance having sufficient computational function to execute software needed to establish and use a connection to network 101. Network appliances 107 may comprise workstation and personal computer hardware executing commercial operating systems such as Unix variants, Microsoft Windows, Macintosh OS, and the like. At the same time, some appliances 107 comprise portable or handheld devices using wireless connections through a wireless access provider such as personal digital assistants and cell phones executing operating system software such as PalmOS, WindowsCE, and the like. Moreover, the present invention is readily extended to network devices such as office equipment, vehicles, and personal communicators that make occasional connection through network 101.

Each of the devices shown in FIG. 1 may include memory, mass storage, and a degree of data processing capability sufficient to manage their connection to network 101. The computer program devices in accordance with the present invention are implemented in the memory of the various devices shown in FIG. 1 and enabled by the data processing capability of the devices shown in FIG. 1. In addition to local memory and storage associated with each device, it is often desirable to provide one or more locations of shared storage such as disk farm (not shown) that provides mass storage capacity beyond what an

individual device can efficiently use and manage. Selected components of the present invention may be stored in or implemented in shared mass storage.

One feature of the present invention is that front-end servers 201 (shown in Fig. 2B) and/or intermediate servers 206 (shown in Fig. 2A) are implemented as an interchangeable pool of servers, any one of which may be dynamically configured to receive request/response traffic of particular clients 205 and servers 210-212. The embodiments of FIG. 2A and FIG. 2B are not strictly alternative as they may coexist in a network environment. A redirection mechanism, shown in Fig. 3, is enabled to select from an available pool of front-end servers 201 and intermediate servers 206 and direct client request packets from the originating web server to a selected front-end server 201 or intermediary server 206.

In the case of web-based environments, front-end 201 is implemented using custom or off-the-shelf web server software. Front-end 201 is readily extended to support other, non-web-based protocols, however, and may support multiple protocols for varieties of client traffic. Front-end 201 processes the data traffic it receives, regardless of the protocol of that traffic, to a form suitable for transport by TMP 202 to a back-end 203. Hence, most of the functionality implemented by front-end 201 is independent of the protocol or format of the data received from a client 205. Hence, although the discussion of the exemplary embodiments herein relates primarily to front-end 201 implemented as a web server, it should be noted that, unless specified to the contrary, web-based traffic management and protocols are merely examples and not a limitation of the present invention.

In the embodiment of Fig. 2A, intermediary servers 206 interact directly with server(s) 210-212. In the embodiment of Fig. 2B, intermediary server 206 is implemented as front-end computer 201 and a back-end computer 203. Front-end server 201 establishes and maintains an enhanced communication channel with a back-end server 203. In either embodiment, intermediary server 206, front-end 201 and/or back-end 203 operate to cache response traffic flowing between a server 210-212 and a client 205.

In the specific examples herein client 205 comprises a network-enabled graphical user interface such as a World Wide Web ("web") browser. However, the present invention is readily extended to client software other than conventional World Wide Web browser software. Any client application that can access a standard or proprietary user level protocol for network access is a suitable equivalent. Examples include client applications that act as front ends for file transfer protocol (FTP) services, voice over Internet protocol (VoIP) services, network news protocol (NNTP) services, multi-purpose internet mail extensions (MIME) services, post office protocol (POP) services, simple mail transfer protocol (SMTP) services, as well as Telnet services. In addition to network protocols, the client application may serve as a database management system (DBMS) in which case the client application generates query language (e.g., structured query language or "SQL") messages. In wireless appliances, a client application functions as a front-end to a wireless access protocol (WAP) service.

Fig. 2B illustrates an embodiment in which intermediary server 206 is implemented by cooperative action of a front-end computer 201 and a back-end computer

203. Caching processes are performed by front-end 201, back-end 203, or both. Front-end mechanism 201 serves as an access point for client-side communications. In one example, front-end 201 comprises a computer that sits
5 "close" to clients 205. By "close", "topologically close" and "logically close" it is meant that the average latency associated with a connection between a client 205 and a front-end 201 is less than the average latency associated with a connection between a client 205 and servers 210-
10 212. Desirably, front-end computers have as fast a connection as possible to the clients 205. For example, the fastest available connection may be implemented in point of presence (POP) of an Internet service provider (ISP) 106 used by a particular client 205. However, the
15 placement of the front-ends 201 can limit the number of browsers that can use them. Because of this, in some applications it may be more practical to place one front-end computer in such a way that several POPs can connect to it. Greater distance between front-end 201 and clients
20 205 may be desirable in some applications as this distance will allow for selection amongst a greater number front-ends 201 and thereby provide significantly different routes to a particular back-end 203. This may offer benefits when particular routes and/or front-ends become
25 congested or otherwise unavailable.

Transport mechanism 202 is implemented by cooperative actions of the front-end 201 and back-end 203. Back-end 203 processes and directs data communication to and from server(s) 210-212. Transport mechanism 202 communicates
30 data packets using a proprietary protocol over the Internet infrastructure in the particular example. Hence, the present invention does not require heavy infrastructure investments and automatically benefits from improvements implemented in the general-purpose network

101. Unlike the general-purpose Internet, front-end 201 and back-end 203 are programmably assigned to serve accesses to a particular server 210-212 at any given time.

It is contemplated that any number of front-end and back-end mechanisms may be implemented cooperatively to support the desired level of service required by the data server owner. The present invention implements a many-to-many mapping of front-ends to back-ends. Because the front-end to back-end mappings can be dynamically changed, a fixed hardware infrastructure can be logically reconfigured to map more or fewer front-ends to more or fewer back-ends and web sites or servers as needed.

A particular advantage of the architectures shown in FIG. 2A and FIG. 2B is that they are readily scaled. In accordance with the present invention, not only can the data itself be distributed, but the functionality and behavior required to implement dynamic content (e.g., dynamic web pages) is readily and dynamically ported to any of a number of intermediary computers 206 and/or front-ends 201 and/or back-ends 203. In this manner, any number of client machines 205 may be supported. To avoid congestion, additional front-ends 201 and/or intermediary servers 206 may be implemented or assigned to particular servers 210-212. Each front-end 201 and/or intermediary server 206 is dynamically re-configurable by updating address parameters to serve particular web sites. Client traffic is dynamically directed to available front-ends 201 to provide load balancing.

In the examples, dynamic configuration is implemented by a front-end manager component 207 (shown only in FIG. 2B) that communicates with multiple front-ends 201 and/or intermediary servers 206 to provide administrative and configuration information to front-ends 201. Each front-

end 201 includes data structures for storing the configuration information, including information identifying the IP addresses of servers 210-212 to which they are currently assigned. Other administrative and configuration information stored in front-end 201 and/or intermediary servers 206 may include information for prioritizing data from and to particular clients, quality of service information, and the like.

Similarly, additional back-ends 203 can be assigned to a web site to handle increased traffic. Back-end manager component 209 couples to one or more back-ends 203 to provide centralized administration and configuration service. Back-ends 203 include data structures to hold current configuration state, quality of service information and the like. In the particular examples front-end manager 207 and back-end manager 209 serve multiple servers 210-212 and so are able to manipulate the number of front-ends and back-ends assigned to a particular server (e.g., server 210) by updating this configuration information. When the congestion for the server 210 subsides, the front-end 201, back-end 203, and/or intermediary server 206 may be reassigned to other, busier servers. These and similar modifications are equivalent to the specific examples illustrated herein.

In order for a client 205 to obtain service from a front-end 201 or intermediate server 206, it must first be directed to a front-end 201 or intermediate server 206. Preferably, client 205 initiates all transactions as if it were contacting the originating server 210-212. FIG. 3 illustrates a domain name server (DNS) redirection mechanism that illustrates how a client 205 is connected to a front-end 201. The DNS systems is defined in a variety of Internet Engineering Task Force (IETF)

documents such as RFC0883, RFC 1034 and RFC 1035 which are incorporated by reference herein. In a typical environment, a client 205 executes a browser 301, TCP/IP stack 303, and a resolver 305. For reasons of performance and packaging, browser 301, TCP/IP stack 303 and resolver 305 are often grouped together as routines within a single software product.

Browser 301 functions as a graphical user interface to implement user input/output (I/O) through monitor 311 and associated keyboard, mouse, or other user input device (not shown). Browser 301 is usually used as an interface for web-based applications, but may also be used as an interface for other applications such as email and network news, as well as special-purpose applications such as database access, telephony, and the like. Alternatively, a special-purpose user interface may be substituted for the more general-purpose browser 301 to handle a particular application.

TCP/IP stack 303 communicates with browser 301 to convert data between formats suitable for browser 301 and IP format suitable for Internet traffic. TCP/IP stack also implements a TCP protocol that manages transmission of packets between client 205 and an Internet service provider (ISP) or equivalent access point. IP protocol requires that each data packet include, among other things, an IP address identifying a destination node. In current implementations the IP address comprises a 32-bit value that identifies a particular Internet node. Non-IP networks have similar node addressing mechanisms. To provide a more user-friendly addressing system, the Internet implements a system of domain name servers that map alpha-numeric domain names to specific IP addresses. This system enables a name space that is more consistent

reference between nodes on the Internet and avoids the need for users to know network identifiers, addresses, routes and similar information in order to make a connection.

5 The domain name service is implemented as a distributed database managed by domain name servers (DNSs) 307 such as DNS_A, DNS_B and DNS_C shown in FIG. 3. Each DNS relies on <domain name:IP> address mapping data stored in master files scattered through the hosts that use the domain system. These master files are updated by local system administrators. Master files typically comprise text files that are read by a local name server, and hence become available through the name servers 307 to users of the domain system.

15 The user programs (e.g., clients 205) access name servers through standard programs such as resolver 305. Resolver 305 includes an address of a DNS 307 that serves as a primary name server. When presented with a reference to a domain name for a data server 210-212, resolver 305 sends a request to the primary DNS (e.g., DNS_A in FIG. 3). The primary DNS 307 returns either the IP address mapped to that domain name, a reference to another DNS 307 which has the mapping information (e.g., DNS_B in FIG. 3), or a partial IP address together with a reference to another DNS that has more IP address information. Any number of DNS-to-DNS references may be required to completely determine the IP address mapping.

30 In this manner, the resolver 305 becomes aware of the IP address mapping which is supplied to TCP/IP component 303. Client 205 may cache the IP address mapping for future use. TCP/IP component 303 uses the mapping to supply the correct IP address in packets directed to a

particular domain name so that reference to the DNS system need only occur once.

In accordance with the present invention, at least one DNS server 307 is owned and controlled by system components of the present invention. When a user accesses a network resource (e.g., a database), browser 301 contacts the public DNS system to resolve the requested domain name into its related IP address in a conventional manner. In a first embodiment, the public DNS performs a conventional DNS resolution directing the browser to an originating server 210-212 and server 210-212 performs a redirection of the browser to the system owned DNS server (i.e., DNS_C in FIG. 3). In a second embodiment, domain:address mappings within the DNS system are modified such that resolution of the originating server's domain automatically return the address of the system-owned DNS server (DNS_C). Once a browser is redirected to the system-owned DNS server, it begins a process of further redirecting the browser 301 to the best available front-end 201.

Unlike a conventional DNS server, however, the system-owned DNS_C in FIG. 3 receives domain:address mapping information from a redirector component 309. Redirector 309 is in communication with front-end manager 207 and back-end manager 209 to obtain information on current front-end and back-end assignments to a particular server 210. A conventional DNS is intended to be updated infrequently by reference to its associated master file. In contrast, the master file associated with DNS_C is dynamically updated by redirector 309 to reflect current assignment of front-end 201 and back-end 203. In operation, a reference to data server 210-212 may result in an IP address returned from DNS_C that points to any

selected front-end 201 that is currently assigned to data server 210-212. Likewise, data server 210-212 can identify a currently assigned back-end 203 by direct or indirect reference to DNS_C.

5 Despite the efficiency of the mechanisms shown in Fig. 3, redirection does take some time and it is preferable to send subsequent requests for a particular server 210-212 directly to an assigned front-end 201 or intermediary server 206 without redirection. When a web
10 page includes links with absolute references the browser 301 may attempt DNS resolution each time a link is followed. To prevent this, one embodiment of the present invention rewrites these links as a part of its reformatting process. In this manner, even though the
15 server contains only a page with absolute references, the page delivered to a client contains relative references.

FIG. 4A illustrates a first embodiment in which a single intermediary computer 206 is used, whereas FIG. 4B and FIG. 5 illustrate a second embodiment where both
20 front-end 201 and back-end 203 are used to implement the intermediary server 206. In the embodiment of FIG. 4A, the intermediary server 206 may be located topologically near the client 205 or data server 210-212--either alternative provides some advantage and the choice of
25 location is made to meet the needs of a particular application. Like identified components are substantially equivalent in Fig. 4A, Fig. 4B and Fig. 5 and for ease of understanding are not duplicatively described herein. Also, the components shown in Fig. 4A and Fig. 4B are
30 optimized for web-based applications. Appropriate changes to the components and protocols are made to adapt the specific examples to other protocols and data types.

Requests from client 205 are received by a TCP unit 401. TCP component 401 includes devices for implementing physical connection layer and Internet protocol (IP) layer functionality. Current IP standards are described in IETF documents RFC0791, RFC0950, RFC0919, RFC0922, RFC792, RFC1112 that are incorporated by reference herein. For ease of description and understanding, these mechanisms are not described in great detail herein. Where protocols other than TCP/IP are used to couple to a client 205, TCP component 401 is replaced or augmented with an appropriate network protocol process.

TCP component 401 communicates TCP packets with one or more clients 205. Preferably, TCP component 401 creates a socket for each request, and returns a received response through the same socket. Received packets are coupled to parser 402 where the Internet protocol (or equivalent) information is extracted. TCP is described in IETF RFC0793 which is incorporated herein by reference. Each TCP packet includes header information that indicates addressing and control variables, and a payload portion that holds the user-level data being transported by the TCP packet. The user-level data in the payload portion typically comprises a user-level network protocol datagram.

Parser 402 analyzes the payload portion of the TCP packet. In the examples herein, HTTP is employed as the user-level protocol because of its widespread use and the advantage that currently available browser software is able to readily use the HTTP protocol. In this case, parser 402 comprises an HTTP parser. More generally, parser 402 can be implemented as any parser-type logic implemented in hardware or software for interpreting the contents of the payload portion. Parser 402 may implement

file transfer protocol (FTP), mail protocols such as simple mail transport protocol (SMTP), structured query language (SQL), and the like. Any user-level protocol, including proprietary protocols, may be implemented within the present invention using appropriate modification of parser 402.

In accordance with the present invention, intermediary 206 and front-end 201 include a caching mechanism 403. Cache 403 may be implemented as a passive cache that stores frequently and/or recently accessed content such as pages or page elements. Cache 403 can also be implemented as an active cache that stores content according to specified rules including pages and/or page elements that have not been requested but that are anticipated to be accessed.

Upon receipt of a TCP packet, HTTP parser 402 determines if the packet is making a request for content within cache 403. If the request can be satisfied from cache 403, the data is supplied directly without reference to server 210-212 (i.e., a cache hit). Cache 403 implements any of a range of management functions for maintaining fresh content. For example, cache 403 may invalidate portions of the cached content after an expiration period specified with the cached data or by sever 210-212. Also, cache 403 may proactively update expired cache contents before a request is received for particularly important or frequently used data from data server 210-212 by requesting updated files from the back-end 203 and/or web server 210-212. Further, cache 403 may be proactively served with updated files(s) by back-end 203 and/or web server 210-212. Cache 403 evicts information using any desired algorithm such as least recently used, least frequently used, first in/first out,

or random eviction. When the requested data is not within
cache 403, a request is passed to server 210-212, and the
returned data may be stored in cache 403. Some requests
must be supplied to server 210-212 (e.g., customer credit
5 information, form data and the like).

Although the present invention may use conventional
cache instructions provided in HTTP headers, preferably
the content includes caching parameters that provides
independent instruction to the caching system of the
10 present invention. In a particular example, each content
element of a web site is associated with a data structure
that specifies a cache expiration date value and a cache
update interval value. The expiration date value
indicates a specific date that the associated content
15 element will expire. The date can be represented, for
example, in seconds since January 1, 1970. This member
will be zero by default. The update interval value
indicates a time interval before the associate content
will expire. The interval is expressed in seconds and has
20 a zero value by default in the particular example.

Because the invention provides for passing cache
instructions that are independent of any instructions in
the HTTP header, the caches of the present invention may
provide caching services even where the content is marked
25 uncacheable in the HTTP header, or where the cache
interval or expiration values associated with a particular
piece of content differ from that specified in the HTTP
header. Moreover, the present invention is readily
extensible to non-HTTP protocols and languages that do not
30 explicitly provide for any cache behavior such as, for
example, FTP, structured query language (SQL), mail
protocols and the like.

In response to a cached content element passing its expiration date or expiration interval, the content may either be removed from the cache or updated by reference to an origin server or other cache that has fresher content. This update can be scheduled immediately or delayed to a more convenient time (e.g. when traffic volume is low).

Preferably, the cache contents are determinable by the site owner by allowing the site owner to cause the cache to be loaded with desired content. Several mechanisms are available to implement this functionality. In one alternative, front-end manager can explicitly load content into cache 403 as desired by a site owner. In another alternative, the cache instructions passed with a response packet may indicate subsequent material to be added to cache 403. For example, when an HTML page is loaded the cache instructions may indicate that all linked resources referenced in the HTML page should be loaded into cache 403. In a more aggressive example, when any one page of a web site 210-212 is loaded, the entire web site is transferred to cache 403 while the user views the first accessed web page. In yet another example, cache contents are propagated by a back-end 203 to connected front-ends 201 based upon propagation rules stored and implemented by back-end 203. For example, when the load on a back-end 203 is high and/or the resources of a back-end 203 become limited, the back-end can more aggressively or speculatively move content out to front-ends 201 to prevent hits. These cache functions may occur as a result of explicit cache instructions, or may be automatically performed by front-end 201 and/or intermediary 206 based on rules (e.g., site owner specified rules) stored in the server.

5 Caching can be performed on a file basis, block
basis, or any other unit of data regardless of its size.
Irrespective of the size of the data unit in cache, cache
content is kept current by, for example, causing the
front-end 201 or intermediary server 206 to check cache
contents against a standard, such as a version of the data
unit that is maintained in origin server 210. When a
difference between the cached data unit and the standard
is detected, the cached unit may be updated by copying the
10 changed data unit, or changed portion of the data unit,
from the standard to cache 403. By updating only changed
material, rather than updating based upon fixed or static
expiration times, data transfers may be lessened in
frequency and quantity, especially for slow changing data.

15 Although back-end cache 503 is discussed below with
reference to Fig. 5, it is important to note that any of
the cache loading operations described above for cache 403
may be implemented using back-end cache 503. For example,
when a user accesses a first page of a web site, some or
20 all of the web site may be loaded into back-end cache 503
rather than cache 403. As front-end cache 403 will
typically contain content from multiple web sites, it may
be burdensome to load cache 403 immediately. Instead,
content can be pushed out from cache 503 to cache 403 over
25 time as the user continues to access the web site. In
this manner, a web site owner can explicitly prevent hits
to the central server(s) by proactively sending content to
the various levels of cache. Similarly, either cache may
proactively or speculatively fill its cache by generating
30 requests to server 210

The default home page or "index.html" page of the web
site (i.e., the page most likely to be the first contact
with a user) may be cached on the intermediary or until

the website provides an updated index page permanently. A hit to the index page will result in serving the index page from the cache, and a request from the intermediary server to load the all or part of the site into the cache

5 403 while the user views the index.html page.

In the case of intermediary server 206, a request that cannot be satisfied from cache 403 is generally passed to transport component 409 for communication to server 210-212 over channel 411. An alternative to this

10 operation is to determine from cache 403 whether an overlapping request is already pending at the server 210-212 for the same content. An overlapping request refers to a request that will generate a response having at least some content elements as a request currently being served.

15 In this circumstance, subsequently arriving requests can be queued or buffered until the first request returns from the server 210-212 and cache 403 is loaded, as the original request is likely to be responded to prior to any subsequent requests for the same information. This may

20 offer valuable performance improvements in situations where, in a particular example, a large file, such as a multimedia file, being accessed nearly simultaneously by a number of users would only be requested once from the back-end 203 or server 210. By "substantially

25 simultaneously" it is meant that many requests for the same file or resource are received prior to receipt of the response from the web server 210 or back-end 203 to the original request.

In an optional implementation, caches 403 and/or 503

30 include an Internet cache protocol (ICP) port for connection to each other and external cache mechanisms. ICP is described in IETF RFC 2186 as a protocol and message format used for communicating between web caches

to locate specific objects in neighboring caches. Essentially, one cache sends an ICP query to a set of caches within neighboring front-ends 201, back-ends 203 and/or intermediary servers 206. The neighboring caches
5 403 and 503 respond back with a "HIT" or "MISS" message indicating whether the requested content exists in the cache.

In a particular embodiment, transport component 409 implements a TCP/IP layer suitable for transport over the
10 Internet or other IP network. Transport component 409 creates a socket connection for each request that corresponds to the socket created in transport component 401. This arrangement enables responses to be matched to requests that generated the responses. Channel 411 is
15 compatible with an interface to server 210-212 which may include Ethernet, Fibre channel, or other available physical and transport layer interfaces.

In Fig. 4A, server 210-212 returns responses to transport component 409 and supplies responses to parser
20 402. Parser 402 implements similar processes with the HTTP response packets as described hereinbefore with respect to request packets.

HTTP component 406 reassembles the response into a format suitable for use by client 205, which in the
25 particular examples herein comprises a web page transported as an HTTP packet. The HTTP packet is sent to transport component 401 for communication to client 205 on the socket opened when the corresponding request was received. In this manner, from the perspective of client
30 205, the request has been served by originating server 210-212.

In the embodiment of Fig. 4A and Fig. 5, intermediary server 206 shown in Fig. 2A is implemented by front-end computer 201 and back-end computer 203. A front-end computer 201 refers to a computer located at the client side of network 101 whereas a back-end computer 203 refers to a computer located at the server side of network 101. This arrangement enables caching to be performed at either or both computers. Hence, in addition to caching data to serve needs of clients 205 and servers 210-212, data can be cached to help regulate transport across the communication link 202 coupling front-end 201 and back-end 203. For example, back-end 203 can function as an active "look-ahead" cache to fill cache 503 with content from a server 210-212 aggressively according to criteria specified by the site owner, whereas front-end can cache more passively by caching only responses to actual requests. Overall performance is improved while controlling the load on TMP connection 202 and on server 210-212.

Optionally, front-end 201, back end 203, and intermediary computer 206 implement security processes, compression processes, encryption processes and the like to condition the received data for improved transport performance and/or provide additional functionality. These processes may be implemented within any of the functional components shown in Fig. 4A, Fig. 4B and Fig. 5 or implemented as separate functional components within front-end 201, back-end 203 or intermediary 206.

In the embodiment of FIG. 4B and FIG. 5, the front-end 201 and back-end 202 are coupled by an enhanced communication channel (e.g., TMP link 202). Blenders 404 and 504 slice and/or coalesce the data portions of the received packets into more desirable "TMP units" that are

5 sized for transport through the TMP mechanism 202. The data portion of TCP packets may range in size depending on client 205 and any intervening links coupling client 205 to TCP component 401. Moreover, where compression or other reformatting is applied, the data will vary in size depending on the reformatting processes. Data blender 404 receives information from front-end manager 207 that enables selection of a preferable TMP packet size. Alternatively, a fixed TMP packet size can be set that yields desirable performance across TMP mechanism 202. Data blenders 404 and 504 also mark the TMP units so that they can be re-assembled at the receiving end.

15 Data blender 404 also serves as a buffer for storing packets from all clients 205 that are associated with front-end 201. Similarly, data blender 504 buffers response packets destined for all the clients 205. Blender 404 mixes data packets coming into front-end 201 into a cohesive stream of TMP packets sent to back-end 203 over TMP link 202. In creating a TMP packet, blender 404 is able to pick and choose amongst the available requests so as to prioritize some requests over others. Prioritization is effected by selectively transmitting request and response data from multiple sources in an order determined by a priority value associated with the particular request and response. For purposes of the present invention, any algorithm or criteria may be used to assign a priority.

30 TMP mechanisms 405 and 505 implement the transport transport morphing protocolTM (TMPTM) packets used in the system in accordance with the present invention. Transport morphing protocol and TMP are trademarks or registered trademarks of Circadence corporation in the United States and other countries. TMP is a TCP-like

protocol adapted to improve performance for multiple channels operating over a single connection. Front-end TMP mechanism 405 in cooperation with a corresponding back-end TMP mechanism 505 shown in FIG. 5 are computer processes that implement the end points or sockets of TMP link 202. The TMP mechanism in accordance with the present invention creates and maintains a stable connection between two processes for high-speed, reliable, adaptable communication.

Another feature of TMP is its ability to channel numerous TCP connections through a single TMP pipe 202. The environment in which TMP resides allows multiple TCP connections to occur at one end of the system. These TCP connections are then combined into a single TMP connection. The TMP connection is then broken down at the other end of the TMP pipe 202 in order to traffic the TCP connections to their appropriate destinations. TMP includes mechanisms to ensure that each TMP connection gets enough of the available bandwidth to accommodate the multiple TCP connections that it is carrying.

An advantage of TMP as compared to traditional protocols is the amount of information about the quality of the connection that a TMP connection conveys from one end to the other of a TMP pipe 202. As often happens in a network environment, each end has a great deal of information about the characteristics of the connection in one direction, but not the other. By knowing about the connection as a whole, TMP can better take advantage of the available bandwidth.

In a particular example, each TMP packet includes a header portion in which some area is available for transfer of commands between a back-end 203 and a front-end 201. For example, a "Fill_Cache" command can be

defined having a parameter specifying a URL having content that is to be loaded into the cache of the recipient of the command. In this manner, a front-end 201 can explicitly cause a back-end to fill its cache with specified content without requiring the content to be forwarded over the TMP link. Similarly, a back-end 203 can send explicit cache fill instructions to a front-end 201. Alternatively, front-end 201 and back-end 201 can communicate cache fill instructions through management components 207 and 209. In yet another alternative, cache fill procedures can be implemented by remote procedure calls (RPCs). For example, a back-end 203 can issue a cache fill RPC to a front-end 201 that when executed on front-end 201 will cause front-end 201 to request the specified content even though no client request has been received. The passive caching mechanism in front-end 201 will then perform the desired cache fill operation on the returned data.

Although the invention has been described and illustrated with a certain degree of particularity, it is understood that the present disclosure has been made only by way of example, and that numerous changes in the combination and arrangement of parts can be resorted to by those skilled in the art without departing from the spirit and scope of the invention, as hereinafter claimed. For example, while devices supporting HTTP data traffic are used in the examples, the HTTP devices may be replaced or augmented to support other public and proprietary protocols including FTP, NNTP, SMTP, SQL and the like. In such implementations the front-end 201 and/or back end 203 are modified to implement the desired protocol. Moreover, front-end 201 and back-end 203 may support different protocols such that the front-end 201 supports, for example, HTTP traffic with a client and the back-end

supports a DBMS protocol such as SQL. Such
implementations not only provide the advantages of the
present invention, but also enable a client to access a
rich set of network resources with minimal client
5 software.